An efficient second-order cone programming approach for optimal selection in tree breeding

Makoto Yamashita [1], Tim J. Mullin [2] and Sena Safarina [3]
Submitted: June 15, 2015.

Abstract:

An important problem in tree breeding is optimal selection from candidate pedigree members to produce the highest performance in seed orchards, while conserving essential genetic diversity. The most beneficial members should contribute as much as possible, but such selection of orchard parents would reduce performance of the orchard progeny due to serious inbreeding. To avoid inbreeding, we should include a constraint on the numerator relationship matrix to keep a group coancestry under an appropriate threshold. Though an SDP (semidefinite programming) approach proposed by Pong-Wong and Woolliams gave an accurate optimal value, it required rather long computation time.

In this paper, we propose an SOCP (second-order cone programming) approach to reduce this computation time. We demonstrate that the same solution is attained by the SOCP formulation, but requires much less time. Since a simple SOCP formulation is not much more efficient compared to the SDP approach, we exploit a sparsity structure of the numerator relationship matrix, and formulate the SOCP constraint using Henderson's algorithm. Numerical results show that the proposed SOCP approach reduced computation time in a case study from 39,200 seconds under the SDP approach to less than 2 seconds.

Keywords: Semidefinite programming, Second-order cone programming, Tree breeding, Optimal selection, Group coancestry, Relatedness, Genetic gain
AMS classification: 90C22 Semidefinite programming, 90C25 Convex programming, 92-08 Biology and other natural sciences (Computational methods).

1 Introduction

The usage of mathematical optimization approaches for tree breeders have been increasing [1, 15, 20, 23], since one of their purposes is to derive better performance from seed orchards. When tree breeders make a plan for new seed orchards, they determine the contributions of candidate pedigree members so that the resultant orchard maximizes response to the selection. From the viewpoint of mathematical optimization, the simplest form of the optimal selection problem is:

$$\begin{array}{lcl} \max & : & \boldsymbol{g}^T\boldsymbol{x} \\ \text{subject to} & : & \boldsymbol{e}^T\boldsymbol{x} = 1 \\ & & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \end{array}$$

[1] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1-W8-29 Ookayama, Meguro-ku, Tokyo 152-8552, Japan (Makoto.Yamashita@is.titech.ac.jp).
[2] The Swedish Forestry Research Institute (Skogforsk), Box 3, Sävar 918 21, Sweden; and 224 rue du Grand-Royal Est, QC, J2M 1R5, Canada.
[3] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1-W8-29 Ookayama, Meguro-ku, Tokyo 152-8552, Japan.

Here, we assume that the number of the candidate members is m. The variable vector is $\boldsymbol{x} \in \mathbb{R}^m$, and corresponds to the contributions of the candidates. The first constraint $\boldsymbol{e}^T\boldsymbol{x} = 1$ indicates that the total contribution of candidate members is unity. We use the vector $\boldsymbol{e} \in \mathbb{R}^m$ to denote the vector of all ones, and the superscript T to denote the transpose of a vector or a matrix. In the second constraints, $\boldsymbol{l} \in \mathbb{R}^m$ and $\boldsymbol{u} \in \mathbb{R}^m$ are element-wise lower and upper bounds on the contributions, respectively. The performance measure appears in the objective function as its coefficient $\boldsymbol{g} = (g_1, \ldots, g_m)^T$, and the estimated breeding value (EBV) [13] is often employed for $\boldsymbol{g}$. The values $g_1, \ldots, g_m$ are calculated separately before we solve the optimal selection problem, so we can consider $g_1, \ldots, g_m$ as constant values.

The above problem is a simple linear programming problem, therefore, a greedy method is enough to solve it. Such solution includes the candidates corresponding to the highest EBV as much as possible, and it is most efficient in situations where all the pedigrees are independent. Even if the pedigrees are independent, diversity is still an issue. Lindgren *et al.* [11] discussed a linear deployment in which the contributions of the candidate members are proportional to their EBVs. In practical situations, however, we cannot overlook the effects due to the relatedness that accumulates over cycles of breeding the pedigrees.

To reflect the effect of the relatedness, Cockerham [4] extended the definitions of coancestry coefficients in order to include coancestry of a group. The group coancestry on the contributions $\boldsymbol{x}$ is calculated with the formula $\frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{2}$, where $\boldsymbol{A} \in \mathbb{R}^{m\times m}$ is the numerator relationship matrix of Wright [29] (we will review a formula for the numerator relationship matrix in Section 2). Introducing a constraint to keep group coancestry under an appropriate level $\theta \in \mathbb{R}$, Meuwissen [15] proposed a formulation of optimal contributions:

$$\begin{array}{lll} \max & : & \boldsymbol{g}^T\boldsymbol{x} \\ \text{subject to} & : & \boldsymbol{e}^T\boldsymbol{x} = 1 \\ & & \frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{2} \leq \theta \\ & & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \end{array} \tag{1}$$

Meuwissen developed an iterative method based on Lagrangian multipliers to solve this optimization problem, and his method has been used in breedings, for example, [6, 9, 28]. It is a characteristic of this method that some variables x_i may be fixed to its lower or upper bounds (l_i or u_i) during the iterations, and Pong-Wong and Woolliams [20] demonstrated that the method did not always obtain the optimal solution.

Pong-Wong and Woolliams utilized the structure of the numerator relationship matrix $\boldsymbol{A}$ to formulate the problem (1) into a semidefinite programming (SDP) problem. SDP is a convex optimization problem that maximizes a linear objective function over the constraints described as linear matrix inequalities. The research in 1990s, for example [7, 10], extended interior-point methods from linear programming problems to SDP problems. Based on primal-dual interior-point methods, software packages (like SDPA [30], SDPARA [31], SDPA-C [32], SDPT3 [26], and SeDuMi [25]) have been developed for solving SDPs. Using the SDP formulation, Pong-Wong and Woolliams obtained the exact optimal value of (1). The number of candidate members discussed in [20], however, was limited to only small sizes, $m \leq 10$. Ahlinder *et al.* [1] implemented the SDP approach into a software package called OPSEL [17][4] with the help of the latest version of SDPA (a high-performance SDP solver) [30]. They solved large problems ($m \geq 10,000$) that were generated from real Scots pine pedigrees and performance data, and they also focused on flexibility and re-optimization of the SDP approach.

A main obstacle in the numerical tests of [1] was that the SDP approach took rather long computation time. They reported for their case study that it required five-hours of computation

[4] `http://www.skogforsk.se/opsel`

time for a problem of the size $m = 12,000$. Even though the SDP guarantee the optimal solution, the computation time is rather long for operational application and requiring significant truncation of the candidate list prior to optimizing the selection.

In this paper, we propose a second-order cone programming (SOCP) approach. SOCP is a convex optimization that maximizes a linear objective function over second-order cone constraints. SOCP can be considered as a special case of SDP, and can be efficiently solved with interior-point methods in a similar way to SDP [24, 27]. Lobo *et al.* [12] discussed wide-range applications of SOCP, for example, filter design and truss design, and Sasakawa and Tsuchiya applied SOCP to magnetic shield design [22]. Alizadeh and Goldfarb [2] surveyed theoretical and algorithmic aspects of SOCP, and the software packages SDPT3 [26] and SeDuMi [25] can solve not only SDP but also SOCP using the primal-dual interior-point methods. In addition, ECOS [5] was also implemented recently to solve SOCP problems.

We first discuss that the proposed SOCP formulation also attains the optimal solution of (1). Since SOCP is a special case of SDP, we could expect that a simple SOCP formulation would be enough to reduce the computation time. However, preliminary numerical tests showed that the simple formulation did not perform well. We therefore utilize a sparsity embedded in the numerator relationship matrix and establish a more efficient SOCP formulation. Furthermore, we integrate Henderson's algorithm [8] into this formulation. Numerical tests with the data including Scots pine showed that the SOCP formulation with Henderson's algorithm reduced a great amount of computation time. For the case of $m = 10,100$, we attained a speedup of 20,000-times compared to the SDP approach.

The rest of this paper is organized as follows. Section 2 describes the SDP approach of Pong-Wong and Woolliams and discusses a simple SOCP formulation. In Section 3, we propose SOCP formulations and derive an efficient method to solve problem (1) . Section 4 shows the numerical results to verify the computation time reduction for problems of various sizes. Finally, Section 5 gives conclusions and discusses future directions.

2 SDP formulation and simple SOCP formulation

Since a principal characteristic of our problem (1) is determined by the numerator relationship matrix $\boldsymbol{A}$, we first review a formula to evaluate its elements. We then describe the SDP approach of Pong-Wong and Woolliams [20], and compare the performance of the SDP approach and a simple SOCP formulation.

To evaluate the elements of the numerator relationship matrix $\boldsymbol{A}$, we separate the set of pedigree candidate members $\mathcal{P} := \{1, 2, \ldots, m\}$ into the three disjoint groups:

$$\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{P}_2,$$

where

$$\begin{cases} \mathcal{P}_0 &= \{i \in \mathcal{P} : \text{both parents } p(i) \text{ and } q(i) \text{ are unknown}\} \\ \mathcal{P}_1 &= \{i \in \mathcal{P} : \text{one parent } p(i) \text{ is known and the other parent } q(i) \text{ is unknown}\} \\ \mathcal{P}_2 &= \{i \in \mathcal{P} : \text{both parents } p(i) \text{ and } q(i) \text{ are known}\}. \end{cases}$$

Figure 1 gives an example of pedigree with $m = 9$ members and illustrates its genealogical chart. In this example, $\mathcal{P}_0 = \{1, 2\}, \mathcal{P}_1 = \{5\}, \mathcal{P}_2 = \{3, 4, 6, 7, 8, 9\}$, and the parents of the 8th member are $p(8) = 7$ and $q(8) = 6$. We use a convention $p(i) = 0$ or $q(i) = 0$ if the parent $p(i)$ or $q(i)$ is unknown, respectively, and we can assume $i > p(i) \geq q(i)$ for all $i \in \mathcal{P}$ without loss of generality.

pedigree id	parents
1	unknown and unknown
2	unknown and unknown
3	1 and 2
4	1 and 2
5	2 and unknown
6	3 and 4
7	1 and 5
8	6 and 7
9	5 and 7

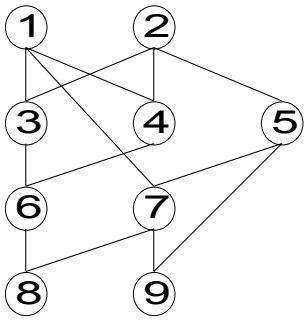

Figure 1: An example of pedigree heredity and its diagram.

The numerator relationship matrix $\boldsymbol{A}$ was defined by Wright [29], and its simplified formula was devised in [8]. The formula of [8] gives the elements $A_{11}, \ldots, A_{nn}$ in a recursive style:

$$\begin{cases} A_{ij} = A_{ji} = \frac{A_{j,p(i)} + A_{j,q(i)}}{2} & \text{for } i = 1, \ldots, m,\ j = 1, \ldots, i-1 \\ A_{ii} = 1 + \frac{A_{p(i),q(i)}}{2} & \text{for } i = 1, \ldots, m, \end{cases}$$

where we use a convention $A_{pq} = 0$ if $p = 0$ or $q = 0$. When we apply this calculation to the example of Figure 1, we obtain the corresponding matrix $\boldsymbol{A}$ as follow:

$$\boldsymbol{A} = \frac{1}{32} \begin{pmatrix} 32 & 0 & 16 & 16 & 0 & 16 & 16 & 16 & 8 \\ 0 & 32 & 16 & 16 & 16 & 16 & 8 & 12 & 12 \\ 16 & 16 & 32 & 16 & 8 & 24 & 12 & 18 & 10 \\ 16 & 16 & 16 & 32 & 8 & 24 & 12 & 18 & 10 \\ 0 & 16 & 8 & 8 & 32 & 8 & 16 & 12 & 24 \\ 16 & 16 & 24 & 24 & 8 & 40 & 12 & 26 & 10 \\ 16 & 8 & 12 & 12 & 16 & 12 & 32 & 22 & 24 \\ 16 & 12 & 18 & 18 & 12 & 26 & 22 & 38 & 17 \\ 8 & 12 & 10 & 10 & 24 & 10 & 24 & 17 & 40 \end{pmatrix}. \tag{2}$$

Pong-Wong and Woolliams [20] formulated the problem (1) into an SDP problem. A standard form of SDP is given by

$$\begin{array}{lcccc} \text{max} & : & & \sum_{k=1}^{m} c_k z_k & \\ \text{subject to} & : & \boldsymbol{F}_0 - & \sum_{k=1}^{m} \boldsymbol{F}_k z_k & \in \mathbb{S}_+^n. \end{array} \tag{3}$$

We use $\mathbb{S}^n$ to denote the space of $n \times n$ symmetric matrices, and $\mathbb{S}_+^n \subset \mathbb{S}^n$ to denote the space of positive semidefinite matrices of dimension n. The variables are $z_1, \ldots, z_m \in \mathbb{R}$, and the input data are $c_1, \ldots, c_m \in \mathbb{R}$ and $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m \in \mathbb{S}^n$.

The key step of Pong-Wong and Woolliams [20] was the usage of the Schur complement of a matrix block. They noticed that the numerator relationship matrix $\boldsymbol{A}$ is always positive definite, and they utilized this property to convert the constraint on the group coancestry to a positive semidefinite condition on the symmetric matrix:

$$\frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{2} \leq \theta \quad \Leftrightarrow \quad \begin{pmatrix} 2\theta & \boldsymbol{x}^T \\ \boldsymbol{x} & \boldsymbol{A}^{-1} \end{pmatrix} \in \mathbb{S}_+^{1+m}. \tag{4}$$

With this positive semidefinite constraint, they converted (1) into the standard SDP form. The input vector $\boldsymbol{c}$ and the input matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ in (3) are given as follow [1, 20]:

$$\boldsymbol{c} \ = \ \boldsymbol{g},$$

Table 1: Computation time on GENCONT and SDP formulation (time in seconds)

m (the number of pedigree)		2,045	10,100
GENCONT	optimal value	438.56	OOM*
	time	67.43	
SDP formulation	optimal value	439.12	47.76
(with 4 cores)	time	70.21	39200.78

*OOM - "out of memory"

$$\boldsymbol{F}_0 = \begin{pmatrix} 1 & & & & & \\ & -1 & & & & \\ & & \mathrm{Diag}(\boldsymbol{u}) & & & \\ & & & -\mathrm{Diag}(\boldsymbol{l}) & & \\ & & & & \begin{pmatrix} 2\theta & \mathbf{0}^T \\ \mathbf{0} & \boldsymbol{A}^{-1} \end{pmatrix} \end{pmatrix},$$

$$\boldsymbol{F}_k = \begin{pmatrix} 1 & & & & & \\ & -1 & & & & \\ & & \mathrm{Diag}(\boldsymbol{e}_i) & & & \\ & & & -\mathrm{Diag}(\boldsymbol{e}_i) & & \\ & & & & \begin{pmatrix} 0 & -\boldsymbol{e}_i^T \\ \boldsymbol{e}_i & \boldsymbol{O} \end{pmatrix} \end{pmatrix} \quad \text{for} \quad k = 1, \ldots, m$$

We use $\boldsymbol{e}_i$ to denote the vector of all zeros except 1 at the ith element, and $\mathrm{Diag}(\boldsymbol{u})$ to denote the diagonal matrix whose diagonal elements are $\boldsymbol{u}$. Note that the dimension of $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ is $3m + 3$. The software package OPSEL [17] automatically formulates the optimal selection into an SDP problem of this form.

Table 1 shows the computed optimal values and the computation time of Meuwissen's implementation (GENCONT) [16] and the SDP formulation. We executed the numerical tests using Matlab R2015a on Windows 8.1 PC with Xeon CPU E3-1231 (3.40 GHz, 4 cores) and 8 GB memory space. We used Windows, since GENCONT can run only on Windows. To solve the SDP (3), we employed SDPA [30].

We observe from Table 1 that the SDP formulation attained a better optimal value than GENCONT. Actually, as shown in [20], the optimal value of the SDP formulation was the exact optimal value, while the Lagrangian multiplier method implemented in GENCONT could not guarantee the optimality. For the large problem ($m = 10,100$), GENCONT gave up the computation, but the SDP formulation again obtained the optimal solution. On the other hand, the disadvantage of the SDP formulation is its computation time. Even using the parallel computing implemented in SDPA, the SDP formulation was slower than GENCONT for $m = 2,045$. Furthermore, the computation time for the large problem exceeded 10 hours even with four cores. When we used only one core, the SDP formulation would require longer computation time than 24 hours.

To reduce the heavy computation time of the SDP formulation, we review the group coancestry constraint $\frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{2} \leq \theta$. With the positive definiteness of $\boldsymbol{A}$, we focus on the property that this constraint can also be described as a second-order condition. The vector $\boldsymbol{v} \in \mathbb{R}^{n_q}$ is said to satisfy the second-order condition if $\boldsymbol{v} \in \mathcal{K}^{n_q}$. The symbol $\mathcal{K}^{n_q}$ denotes the second-order cone of dimension n_q:

$$\mathcal{K}^{n_q} := \left\{ \boldsymbol{v} \in \mathbb{R}^{n_q} : v_1 \geq \sqrt{\sum_{k=2}^{n_q} v_k^2} \right\}.$$

The second-order cone is a special case of positive semidefinite constraint. In fact, using the Schur complement, we can verify that

$$\boldsymbol{v} \in \mathcal{K}^{n_q} \quad \Leftrightarrow \quad \begin{pmatrix} v_1 & v_2 & v_3 & \cdots & v_n \\ v_2 & v_1 & 0 & \cdots & 0 \\ v_3 & 0 & v_1 & \cdots & 0 \\ \vdots & 0 & \vdots & \ddots & 0 \\ v_n & 0 & 0 & \cdots & v_1 \end{pmatrix} \in \mathbb{S}_+^{n_q}.$$

Furthermore, since the numerator relationship matrix $\boldsymbol{A}$ is positive definite, we can apply the Cholesky factorization to $\boldsymbol{A}$ to obtain the upper triangular matrix $\boldsymbol{U}$ such that $\boldsymbol{A} = \boldsymbol{U}^T\boldsymbol{U}$. This factorization derives a second-order cone condition that corresponds to the group coancestry constraint:

$$\frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{2} \leq \theta \quad \Leftrightarrow \quad \boldsymbol{x}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{x} \leq 2\theta \quad \Leftrightarrow \quad ||\boldsymbol{U}\boldsymbol{x}|| \leq \sqrt{2\theta} \quad \Leftrightarrow \quad \begin{pmatrix} \sqrt{2\theta} \\ \boldsymbol{U}\boldsymbol{x} \end{pmatrix} \in \mathcal{K}^{1+m}. \tag{5}$$

Since the most difficult constraint in (1) can be expressed using a second-order cone, it is natural to consider its SOCP formulation. A standard form of SOCP problem is described by

$$\begin{array}{lcl} \text{max} & : & \boldsymbol{c}^T\boldsymbol{z} \\ \text{subject to} & : & \boldsymbol{f}_0 - \boldsymbol{F}\boldsymbol{z} \in \mathbb{R}_+^{n_l} \times \mathcal{K}^{n_q}. \end{array} \tag{6}$$

We use $\mathbb{R}_+^{n_l} \times \mathcal{K}^{n_q}$ to denote the Cartesian product of the non-negative orthant of dimension n_l, $\mathbb{R}_+^{n_l} := \{\boldsymbol{v} \in \mathbb{R}^{n_l} : v_i \geq 0 \text{ for } i = 1, \ldots, n_l\}$, and the second-order cone of dimension n_q. In this problem, the variable vector is $\boldsymbol{z} \in \mathbb{R}^m$, and the input data are $\boldsymbol{c} \in \mathbb{R}^m, \boldsymbol{f}_0 \in \mathbb{R}_+^{n_l+n_q}$ and $\boldsymbol{F} \in \mathbb{R}^{(n_l+n_q)\times m}$. A more general SOCP form than (6) can be defined so that it can handle the Cartesian product of multiple second-order cones, but one cone is enough for the discussion in this paper.

If we formulate the optimal selection problem (1) in a simple style, we obtain an SOCP problem:

$$\begin{array}{lcl} \text{max} & : & \boldsymbol{g}^T\boldsymbol{x} \\ \text{subject to} & : & \begin{pmatrix} 1 \\ -1 \\ \boldsymbol{u} \\ -\boldsymbol{l} \\ \hline \sqrt{2\theta} \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \boldsymbol{e}^T \\ -\boldsymbol{e}^T \\ \boldsymbol{I} \\ -\boldsymbol{I} \\ \hline 0 \\ \boldsymbol{U} \end{pmatrix} \boldsymbol{x} \in \mathbb{R}_+^{2+2m} \times \mathcal{K}^{1+m}, \end{array} \tag{7}$$

We use $\boldsymbol{I}$ to denote the identity matrix of dimension m. We call this formulation *a simple SOCP formulation*. We emphasize that this simple SOCP formulation also gives the exact optimal value of the optimal selection problem (1) due to the equivalence from (4) and (5);

$$\begin{pmatrix} 2\theta & \boldsymbol{x}^T \\ \boldsymbol{x} & \boldsymbol{A}^{-1} \end{pmatrix} \in \mathbb{S}_+^{1+m} \quad \Leftrightarrow \quad \begin{pmatrix} \sqrt{2\theta} \\ \boldsymbol{U}\boldsymbol{x} \end{pmatrix} \in \mathcal{K}^{1+m}.$$

Since SOCP is a special case of SDP, we expected that we could solve the SOCP formulation (7) faster than the SDP formulation (3). In Table 2, we compare the computation times of the SDP and SOCP formulations. We used ECOS [5] as the SOCP solver for (7). For the small problem ($m = 2,045$), the SOCP formulation successfully reduced the computation time from 70.21 seconds to 0.28 seconds, so its speed-up was 250-times. However, for the large problem ($m = 10,100$), the speed-up was limited to 7-times. When we consider the computational complexity, the simple SOCP formulation would be slower than the SDP formulation for further large problems. On contrary to the fact that SOCP is a special case of SDP, the simple SOCP formulation did not seem promising.

Table 2: Computation time on SDP and simple SOCP formulations (time in seconds)

m (the number of pedigree)	2,045	10,100
SDP formulation (3)	70.21	39200.78
simple SOCP formulation (7)	0.28	5604.25

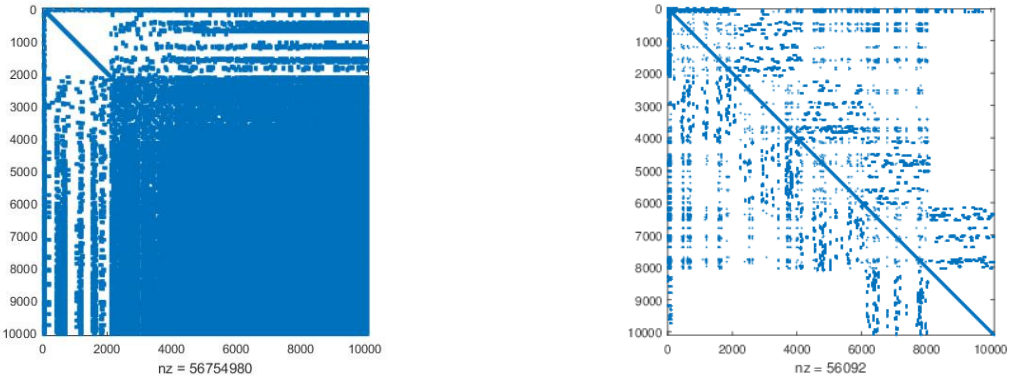

Figure 2: The positions of non-zero elements of $\boldsymbol{A}$ (left) and $\boldsymbol{A}^{-1}$ (right) for the problem of size $m = 10,100$.

3 Efficient formulation based on SOCP

To obtain an efficient formulation based on SOCP, we investigated key properties of the simple SOCP formulation (7). In particular, we focused on the structure of the numerator relationship matrix $\boldsymbol{A}$ and its inverse $\boldsymbol{A}^{-1}$, since the SDP formulation uses only $\boldsymbol{A}^{-1}$. Figure 2 illustrates the positions of the non-zero elements of $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ for the problem of size $m = 10,100$. The dimensions of $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ corresponds to the size m. We can see from Figure 2 that $\boldsymbol{A}^{-1}$ is much sparser than $\boldsymbol{A}$. The numbers of non-zero elements in $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ are 56,754,980 and 56,092, respectively, hence their density against the fully-dense matrix (m^2 non-zero elements) are 55.6% and 0.0549%, respectively. When we apply the Cholesky factorization to $\boldsymbol{A}$, the upper-triangular matrix $\boldsymbol{U}$ inherits the dense property. The number of non-zero elements in $\boldsymbol{U}$ is 23,171,296, and its density against the fully-dense upper triangular matrix is 45.4%. We should utilize the property that $\boldsymbol{A}^{-1}$ is remarkably sparse compared to $\boldsymbol{A}$ and $\boldsymbol{U}$.

From this observation, the direction we should pursue is to use $\boldsymbol{A}^{-1}$ instead of $\boldsymbol{A}$ itself. A key step of our approach is to introduce the new variable $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. Then, the replacement $\boldsymbol{x}$ by $\boldsymbol{A}^{-1}\boldsymbol{y}$ in the optimal selection (1) leads to an equivalent optimization problem:

$$\begin{array}{lcl} \max & : & (\boldsymbol{A}^{-1}\boldsymbol{g})^T\boldsymbol{y} \\ \text{subject to} & : & (\boldsymbol{A}^{-1}\boldsymbol{e})^T\boldsymbol{y} = 1 \\ & & \dfrac{\boldsymbol{y}^T\boldsymbol{A}^{-1}\boldsymbol{y}}{2} \leq \theta \\ & & \boldsymbol{l} \leq \boldsymbol{A}^{-1}\boldsymbol{y} \leq \boldsymbol{u}. \end{array}$$

The Cholesky factor of $\boldsymbol{A}^{-1}$ is the transposed matrix of $\boldsymbol{U}^{-1}$, denoted by $\boldsymbol{U}^{-T}$: namely, $\boldsymbol{A}^{-1} = (\boldsymbol{U}^{-T})^T\boldsymbol{U}^{-T}$. In practical implementation, since $\boldsymbol{A}^{-1}$ is remarkably sparse, we apply an appropriate row/column permutation like AMD (approximate minimum degree permutation) [3] to $\boldsymbol{A}^{-1}$ in order to reduce the number of fill-in (the non-zero elements that newly appear in $\boldsymbol{U}^{-T}$ during the process of the Cholesky factorization). Using $\boldsymbol{U}^{-T}$, we convert this problem into an SOCP

Table 3: Performance comparison on the the SDP formulation, the simple SOCP formulation, and the sparse SOCP formulation (time in seconds).

m (size of pedigree) = 2,045				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	24300	0.52	69.55	70.21
simple SOCP formulation (7)	18201	0.10	0.04	0.28
sparse SOCP formulation (8)	30348	0.37	0.05	0.55
m (size of pedigree) = 10,100				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	121703	26.97	39173.03	39200.78
simple SOCP formulation (7)	23231801	15.95	5587.53	5604.25
sparse SOCP formulation (8)	159570	24.14	0.68	25.60

problem:

$$\begin{array}{lll} \text{max} & : & (\boldsymbol{A}^{-1}\boldsymbol{g})^T\boldsymbol{y} \\ \text{subject to} & : & \begin{pmatrix} 1 \\ -1 \\ \boldsymbol{u} \\ -\boldsymbol{l} \\ \hline \sqrt{2\theta} \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} (\boldsymbol{A}^{-1}\boldsymbol{e})^T \\ -(\boldsymbol{A}^{-1}\boldsymbol{e})^T \\ \boldsymbol{A}^{-1} \\ -\boldsymbol{A}^{-1} \\ \hline 0 \\ \boldsymbol{U}^{-T} \end{pmatrix} \boldsymbol{y} \in \mathbb{R}^{2+2m}_+ \times \mathcal{K}^{1+m}, \end{array} \tag{8}$$

We call this formulation *a sparse SOCP formulation* of the optimal selection problem (1).

We use the matrix $\boldsymbol{A}$ to formulate this new SOCP formulation, but we do not have to use $\boldsymbol{A}$ to solve the resultant SOCP problem with the interior-point methods. Furthermore, when we obtain the optimal solution $\boldsymbol{y}^*$ of (8), we can also obtain the optimal solution $\boldsymbol{x}^*$ of the original problem (1) via the relation $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{y}^*$ without using the dense matrix $\boldsymbol{A}$.

In Table 3, we compare the performance of the SDP formulation (3), the simple SOCP formulation (7), and the sparse SOCP formulation (8). In this table, the first column 'nnz' is the total number of non-zero elements of $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ for SDP and that of $\boldsymbol{f}_0$ and $\boldsymbol{F}$ for SOCP. The second column is the computation time to convert the pedigree like Figure 1 and EBVs (estimated breeding values) into the SDP or SOCP formulations, and the third column is the computation time of the solvers. We applied SDPA with four cores to the SDP formulation and ECOS to the SOCP formulations. The fourth (last) column is the total computation time.

Table 3 shows that for the large problem, the computation time of the sparse SOCP formulation was much shorter than that of the simple SOCP formulation. The sparse SOCP formulation seemed to have more complex structure than the simple SOCP formulation since it repeatedly contained $\boldsymbol{A}^{-1}$ in the matrix $\boldsymbol{F}$, but the total number of non-zeros was reduced from 23,231,801 in the simple SOCP formulation to 159,570 in the sparse SOCP formulation. This led the computation time reduction for the SOCP solver.

To reduce the total time further, we now address the computation time to build the SOCP formulation. In the case of $m = 10,100$, the conversion time occupied 94 % of the total time. In particular, the construction of the dense matrix $\boldsymbol{A}$ and its inversion are the principal bottlenecks.

We investigate further the properties of $\boldsymbol{A}^{-1}$, and employ a compact algorithm to construct $\boldsymbol{A}^{-1}$ proposed by Henderson [8]. In the compact algorithm, we use the vector of inbreeding

Table 4: A compact algorithm to obtain the inverse of the numerator relationship matrix

$\boldsymbol{A}^{-1} \leftarrow \boldsymbol{O}$.
for $i = 1, 2, \ldots, m$
  $b_i \leftarrow \frac{4}{(1+\delta(p(i)))(1-h_{p(i)})+(1+\delta(q(i)))(1-h_{q(i)})}$
  if $i \in \mathcal{P}_0$ **then**
    add b_i to A^{-1}_{ii}
  elseif $i \in \mathcal{P}_1$ **then**
    add b_i to A^{-1}_{ii}
    add $-\frac{b_i}{2}$ to $A^{-1}_{i,p(i)}$, and $A^{-1}_{p(i),i}$
    add $\frac{b_i}{4}$ to $A^{-1}_{p(i),p(i)}$
  elseif $i \in \mathcal{P}_2$ **then**
    add b_i to A^{-1}_{ii}
    add $-\frac{b_i}{2}$ to $A^{-1}_{i,p(i)}$, $A^{-1}_{p(i),i}$, $A^{-1}_{i,q(i)}$, and $A^{-1}_{q(i),i}$
    add $\frac{b_i}{4}$ to $A^{-1}_{p(i),p(i)}$, $A^{-1}_{p(i),q(i)}$, $A^{-1}_{q(i),p(i)}$, and $A^{-1}_{q(i),q(i)}$
  endif
endfor

coefficients defined by $\boldsymbol{h} := diag(\boldsymbol{A}) - \boldsymbol{e}$, where $diag(\boldsymbol{A})$ is a vector composed of the diagonal elements of $\boldsymbol{A}$. Quaas [21] devised an efficient method to compute the inbreeding coefficients $\boldsymbol{h}$ without constructing the matrix $\boldsymbol{A}$ itself, and Masuda *et al.* utilized this method to implement their YAMS package [14]. The compact algorithm in [8] with the enhancement [21] is summarized in Table 4. We use an indicator function $\delta(p)$ such that $\delta(0) = 1$, and $\delta(p) = 0$ for $p \neq 0$. We also use the notation A^{-1}_{ij} to denote the (i, j) element of $\boldsymbol{A}^{-1}$.

The compact formula for Table 4 can be described as

$$\begin{aligned}\boldsymbol{A}^{-1} &= \sum_{i\in\mathcal{P}_0} b_i \boldsymbol{e}_i \boldsymbol{e}_i^T + \sum_{i\in\mathcal{P}_1} b_i \left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)}\right)\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)}\right)^T \\ &\quad + \sum_{i\in\mathcal{P}_2} b_i \left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)} - \frac{1}{2}\boldsymbol{e}_{q(i)}\right)\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)} - \frac{1}{2}\boldsymbol{e}_{q(i)}\right)^T.\end{aligned}$$

When we apply this formula to the example in Figure 1, we obtain the inverse of the numerator relationship matrix as follow:

$$\boldsymbol{A}^{-1} = \frac{1}{42}\begin{pmatrix} 105 & 42 & -42 & -42 & 21 & 0 & -42 & 0 & 0 \\ 42 & 98 & -42 & -42 & -28 & 0 & 0 & 0 & 0 \\ -42 & -42 & 105 & 21 & 0 & -42 & 0 & 0 & 0 \\ -42 & -42 & 21 & 105 & 0 & -42 & 0 & 0 & 0 \\ 21 & -28 & 0 & 0 & 98 & 0 & -21 & 0 & -42 \\ 0 & 0 & -42 & -42 & 0 & 108 & 24 & -48 & 0 \\ -42 & 0 & 0 & 0 & -21 & 24 & 129 & -48 & -42 \\ 0 & 0 & 0 & 0 & 0 & -48 & -48 & 96 & 0 \\ 0 & 0 & 0 & 0 & -42 & 0 & -42 & 0 & 84 \end{pmatrix}.$$

We can see in this example that $\boldsymbol{A}^{-1}$ has more zero-elements than $\boldsymbol{A}$ of (2).

It is known that $b_i > 0$ for any $i = 1, 2, \ldots, m$ from properties of the inbreeding coefficients. Therefore, the constraint on the group coancestry can be transformed into another second-order cone constraint:

$$\frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{2} \leq \theta \quad \Leftrightarrow \quad \frac{\boldsymbol{y}^T \boldsymbol{A}^{-1} \boldsymbol{y}}{2} \leq \theta$$

$$\Leftrightarrow \quad \sum_{i\in\mathcal{P}_0} b_i y_i^2 + \sum_{i\in\mathcal{P}_1} b_i \left(y_i - \frac{1}{2} y_{p(i)} \right)^2 + \sum_{i\in\mathcal{P}_2} b_i \left(y_i - \frac{1}{2} y_{p(i)} - \frac{1}{2} y_{q(i)} \right)^2 \leq 2\theta$$

$$\Leftrightarrow \quad \begin{pmatrix} \sqrt{2\theta} \\ \boldsymbol{B}\boldsymbol{y} \end{pmatrix} \in \mathcal{K}^{1+m}. \tag{9}$$

Here, $\boldsymbol{B}$ is the matrix whose ith row vector is defined by

$$B_{i*} = \begin{cases} \sqrt{b_i}\boldsymbol{e}_i^T & \text{for} \quad i \in \mathcal{P}_0 \\ \sqrt{b_i}\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)}\right)^T & \text{for} \quad i \in \mathcal{P}_1 \\ \sqrt{b_i}\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)} - \frac{1}{2}\boldsymbol{e}_{q(i)}\right)^T & \text{for} \quad i \in \mathcal{P}_2. \end{cases}$$

We now replace the matrix $\boldsymbol{U}^{-T}$ in the sparse SOCP formulation (8) by $\boldsymbol{B}$, hence, we derive another SOCP formulation:

$$\begin{array}{lll} \text{max} & : & (\boldsymbol{A}^{-1}\boldsymbol{g})^T\boldsymbol{y} \\ \text{subject to} & : & \begin{pmatrix} 1 \\ -1 \\ \boldsymbol{u} \\ -\boldsymbol{l} \\ \hline \sqrt{2\theta} \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} (\boldsymbol{A}^{-1}\boldsymbol{e})^T \\ -(\boldsymbol{A}^{-1}\boldsymbol{e})^T \\ \boldsymbol{A}^{-1} \\ -\boldsymbol{A}^{-1} \\ \hline 0 \\ \boldsymbol{B} \end{pmatrix} \boldsymbol{y} \in \mathbb{R}_+^{2+2m} \times \mathcal{K}^{1+m}, \end{array} \tag{10}$$

We call this formulation *a compact SOCP formulation.*

The combination of the compact algorithm in Table 4 and the second-order cone constraint (9) gives us a formulation that does not rely on any dense matrices. Table 5 adds the results of the compact SOCP formulation to Table 3. From Table 5, we observe that the solver time for the compact SOCP formulation was slightly shorter than the sparse SOCP formulation. A further computation time reduction was obtained in the computation time to build the SOCP formulations from the pedigree. In the case $m = 10,100$, the conversion was reduced from 24.14 seconds to 0.37 seconds. Furthermore, we saved a lot of memory space. For the case $m = 10,100$, the matrix $\boldsymbol{A}$ required 778 MB of memory, while in contrast the memory required for $\boldsymbol{B}$ is only 2.33 MB.

4 Numerical tests

We conducted a numerical evaluation of the SDP and SOCP formulations on several datasets. The datasets are for problems of sizes 2045, 5050, 15100, 15222, 50100, 100100 and 300100. The data with the sizes 2,045 and 15,222 were from Scots pine orchards and loblolly pine orchards, respectively, and these data are available at the Dryad Digital Repository `http://dx.doi.org/10.5061/dryad.9pn5m`. The other data were generation by simulation of five cycles of breeding in a closed population using the approach of [18, 19].

Although we used Matlab for the comparison between the formulations, we also implemented the compact SOCP formulation with C++. There were two reasons to implement it outside a Matlab environment. The first is that we can directly know the structure of non-zero elements that appear in the matrix $\boldsymbol{B}$. Therefore, a specified data structure can accelerate the computation time to arrange the input data for building $\boldsymbol{F}$. Secondly, we expect a software package that does not depend on commercial software would extend the opportunity for the field application in tree breeding. Due to the latter reason, we also excluded commercial SOCP solvers from the numerical tests.

Table 5: Performance comparison on the the SDP formulation and the SOCP formulations (time in seconds).

m (size of pedigree) = 2,045				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	24300	0.52	69.55	70.21
simple SOCP formulation (7)	18201	0.10	0.04	0.28
sparse SOCP formulation (8)	30348	0.37	0.05	0.55
compact SOCP formulation (10)	30249	0.01	0.05	0.19
m (size of pedigree) = 10,100				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	121703	26.97	39173.03	39200.78
simple SOCP formulation (7)	23231801	15.95	5587.53	5604.25
sparse SOCP formulation (8)	159570	24.14	0.68	25.60
compact SOCP formulation (10)	153001	0.37	0.62	1.76

One of the advantages of the compact SOCP formulation is that we do not need numerical routines for the Cholesky factorization. If we implement the simple or sparse SOCP formulation with C++, we have to embed certain numerical routines for the Cholesky factorization. In addition, the sparse Cholesky factorization requires a preprocessing by AMD [3] to derive its best performance. In contrast, the compact SOCP formulation obtains the matrix directly $\boldsymbol{F}$ as discussed in Section 3.

The computation environment for the small problems ($m \leq 10,100$) was Matlab R2015a on a Windows 8.1 PC with Xeon E3-1231 (3.40 GHz) and 8 GB memory space. For the large problems ($m \geq 15,100$), we used Matlab R2014b on a Debian Linux server with Opteron 4386 (3.10 GHz) and 128 GB memory space, since 8 GB memory space was not enough for the SDP formulation.

Table 6 shows the numerical results of the SDP and SOCP formulations. We observe from this table that the SDP formulation demanded rather long computation time, particularly for the larger problems. For the problem $m = 15,222$, the compact SOCP formulation with C++ reduced the 22,566 seconds of the SDP formulation to only 2.21 seconds.

Another significant advantage of the compact formulation is memory consumption. The SDP formulation consumed 31 GB memory space to solve the problem $m = 15,222$, and it failed to handle $m \geq 50,100$, despite having 128 GB of memory available. If we use a rough estimation, the memory required for the largest problem $m = 300,100$ would be 12,000 GB. The simple SOCP formulation also suffered from a heavy memory requirement to store the dense matrix $\boldsymbol{U}$. The sparse SOCP formulation did not require the dense matrix $\boldsymbol{A}$ in the resultant SOCP problem, but it utilized $\boldsymbol{A}$ and computed $\boldsymbol{A}^{-1}$, hence it required the long conversion time in the same way as the SDP formulation. In contrast, the compact SOCP formulation consumed less than 766 MB memory space to solve even the largest problem $m = 300,100$. This memory reduction was mainly a result of employing Henderson's algorithm.

From the numerical results, we also observe that the compact SOCP formulation with C++ is faster than that with Matlab. The discrepancy between Matlab (99.65 seconds) and C++ (82.41 seconds) in the largest problem was due to a specified data structure written in C++. In particular, the structure was effective when we arranged the pedigree before building $\boldsymbol{F}$.

Table 6: Numerical results on SDP and SOCP formulations (time in seconds).

m (size of pedigree) = 2,045				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	24300	0.52	69.55	70.21
simple SOCP formulation (7)	18201	0.10	0.04	0.28
sparse SOCP formulation (8)	30348	0.37	0.05	0.55
compact SOCP formulation (10)	30249	0.01	0.05	0.20
compact SOCP formulation with C++	28246	0.01	0.06	0.09
m (size of pedigree) = 5,050				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	60853	4.63	887.32	892.30
simple SOCP formulation (7)	6812127	1.60	696.10	698.15
sparse SOCP formulation (8)	78405	3.67	0.19	4.21
compact SOCP formulation (10)	76533	0.04	0.19	0.58
compact SOCP formulation with C++	76533	0.01	0.21	0.28
m (size of pedigree) = 15,100				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	181703	157.41	21836.63	21994.87
simple SOCP formulation (7)	54063065	26.17	38733.01	38760.00
sparse SOCP formulation (8)	234760	145.53	2.13	148.49
compact SOCP formulation (10)	227989	0.04	2.06	2.92
compact SOCP formulation with C++	227989	0.02	1.95	1.99
m (size of pedigree) = 15,222				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)	181947	161.99	22403.30	22566.11
simple SOCP formulation (7)	7889551	17.96	618.18	636.95
sparse SOCP formulation (8)	227758	150.07	2.13	153.01
compact SOCP formulation (10)	227203	0.04	2.20	3.05
compact SOCP formulation with C++	227203	0.02	2.16	2.21
m (size of pedigree) = 50,100				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)				OOM*
simple SOCP formulation (7)				OOM
sparse SOCP formulation (8)	759294	4989.55	7.58	4999.90
compact SOCP formulation (10)	753023	0.15	7.71	10.63
compact SOCP formulation with C++	753023	0.08	7.48	7.69
m (size of pedigrees) = 100,100				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)				OOM
simple SOCP formulation (7)				OOM
sparse SOCP formulation (8)				$>$ 24 hours**
compact SOCP formulation (10)	1502983	0.35	18.44	24.76
compact SOCP formulation with C++	1502983	0.16	17.57	17.92
m (size of pedigrees) = 300,100				
	nnz	time (conversion)	time (solver)	time (total)
SDP formulation (3)				OOM
simple SOCP formulation (7)				OOM
sparse SOCP formulation (8)				OOM
compact SOCP formulation (10)	4503065	1.10	82.41	99.65
compact SOCP formulation with C++	4503065	0.51	78.54	79.62

* OOM = "out of memory"

* $>$ 24 hours = "the computation failed to complete with in in 24 hours."

5 Conclusions and future directions

We examined the SOCP formulations for the optimal selection problem arising from tree breeding. We employed the transformation $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}$ based on the sparsity of $\boldsymbol{A}^{-1}$ and the efficient method to build $\boldsymbol{A}^{-1}$ by the Henderson's algorithm with the Quaas enhancement. The compact SOCP formulation thus did not involve any dense matrix or the Cholesky factorization. The numerical results demonstrated that the compact SOCP formulation obtained the optimal solution significantly faster than the existing SDP formulation.

The SOCP formulations proposed in this paper may look rather simple for researchers in the field of mathematical optimization. However, the computation time reduction in the optimal selection problem will help improve operational application in tree breeding. We expect that this paper will be one of bridges to introduce efficient approaches cultivated in mathematical optimization to tree breeders.

In this paper, we discussed an *unequal* deployment of parental genotypes to seed orchards, where the contributions of selected members are not required to be equal. To deal with *equal* deployment, as might be appropriate for selection of a fixed-size breeding population, we will need a method to solve a mixed integer SOCP problem; that is an SOCP problem in which some variables are constrained to be integers. The structure of the SOCP formulation developed in this paper will be a basis for an efficient method to consider the mixed integer SOCP problem in the optimal selection problems.

Acknowledgments

We are grateful to Dr. Yutaka Masuda of Obihiro Univeristy of Agriculture and Veterinary Medicine for providing access to the source code of the YAMS package. Our work was partially supported by funding from JSPS KAKENHI (Grant-in-Aid for Scientific Research (C), 15K00032) and Föreningen Skogsträdsförädling (The Swedish Tree Breeding Foundation).

References

[1] J. Ahlinder, T. J. Mullin, and M. Yamashita. Using semidefinite programming to optimize unequal deployment of genotypes to a clonal seed orchard. *Tree Genet. Genomes*, 10(1):27–34, 2014.

[2] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Math. Prog. B*, 95(1):3–51, 2003.

[3] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388, 2004.

[4] C. C. Cockerham. Group inbreeding and coancestry. *Genetics*, 56(1):89–104, 1967.

[5] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *Proceedings of European Control Conference*, pages 3071–3076, 2013.

[6] B. Grundy, B. Villanueva, and J. A. Wooliams. Dynamic selection procedures for constrained inbreeding and their consequences for pedigree development. *Genet. Res.*, 72(2):159–168, 1998.

[7] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. Optim.*, 6(2):342–361, 1996.

[8] C. R. Henderson. A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. *Biometrics*, 32(1):69–83, 1976.

[9] D. Hinrichs and T. H. E. Meuwissen. Analyzing the effect of different approaches of penalized relationship in multistage selection schemes. *J. Anim. Sci.*, 89(11):3426–32, 2011.

[10] M. Kojima, S. Shindoh, and S. Hara. Interior-point methods for the monotone semidefinite linear complementarity problems in symmetric matrices. *SIAM J. Optim.*, 7:86–125, 1997.

[11] D. Lindgren, W. S. Libby, and F. L. Bondesson. Deployment to plantations of numbers and proportions of clones with special emphasis on maximizing gain at a constant diversity. *Theor. Appl. Genet.*, 77(6):825–831, 1989.

[12] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebr. Appl.*, 284(1):193–228, 1998.

[13] M. Lynch and B. Walsh B. *Genetics and Analysis of Quantitative Traits*. Sinauer Associates, Inc., Sunderland, MA, USA, 1998.

[14] Y. Masuda, T. Baba, and M. Suzuki. Application of supernodal sparse factorization and inversion to the estimation of (co) variance components by residual maximum likelihood. *J. Anim. Breed. Genet.*, 131(3):227–236, 2014.

[15] T. H. E. Meuwissen. Maximizing the response of selection with a predefined rate of inbreeding. *J. Anim. Sci.*, 75:934–940, 1997.

[16] T. H. E. Meuwissen. GENCONT: an operational tool for controlling inbreeding in selection and conservation schemes. In *Proceeding of 7th World Congress on Genetics Applied to Livestock Production*, 2002.

[17] T. J. Mullin. OPSEL 1.0: A computer program for optimal selection in forest tree breeding by mathematical programming. Technical Report Nr. 841-2014, Arbetsrapport från Skogforsk, 2014.

[18] T. J. Mullin, J. Hallander, O. Rosvall, and B. Andersson. Using simulation to optimise tree breeding programmes in Europe: an introduction to POPSIM. Technical Report Nr. 711-2010, Arbetsrapport från Skogforsk, 2010.

[19] T. J. Mullin and Y. S. Park. Stochastic simulation of population management strategies for tree breeding: a new decision-support tool for personal computers. *Silvae Genetica*, 44(2):132–140, 1995.

[20] R. Pong-Wong and J. A. Woolliams. Optimisation of contribution of candidate parents to maximise genetic gain and restricting inbreeding using semidefinite programming. *Genet. Sel. Evol*, 39:3–25, 2007.

[21] R. L. Quaas. Computing the diagonal elements and inverse of a large numerator relationship matrix. *Biometrics*, pages 949–953, 1976.

[22] T. Sasakawa and T. Tsuchiya. Optimal magnetic shield design with second-order cone programming. *SIAM J. Sci. Comput.*, 24(6):1930–1950, 2003.

[23] S. Schierenbeck, E. Pimentel, M. Tietze, J. Körte, R. Reents, F. Reinhardt, H. Simianer, and S. König. Controlling inbreeding and maximizing genetic gain using semi-definite programming with pedigree-based and genomic relationships. *J. Dairy Sci.*, 94(12):6143–6152, 2011.

[24] S. H. Schmieta and F. Alizadeh. Associative and jordan algebras, and polynomial time interior-point algorithms for symmetric cones. *Math. Oper. Res.*, 26(3):543–564, 2001.

[25] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, 11 & 12(1-4):625–653, 1999.

[26] K. C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 – a MATLAB software package for semidefinite programming, version 1.3. *Optim. Methods Softw.*, 11 & 12(1-4):545–581, 1999.

[27] T. Tsuchiya. A convergence analysis of the scaling-invariant primal-dual path-following algorithms for second-order cone programming. *Optim. Methods Softw.*, 11 & 12(1-4):141–182, 1999.

[28] J. Woolliams. Genetic contributions and inbreeding. In Kor Oldenbroek, editor, *Utilisation and Conservation of Farm Animal Genetic Resources*, pages 147–165. Wageningen Academic Publishers, The Netherlands, 2007.

[29] S. Wright. Coeffcients of inbreeding and relationship. *Am. Nat.*, 56:330–338, 1922.

[30] M. Yamashita, K. Fujisawa, M. Fukuda, K. Kobayashi, K. Nakta, and M. Nakata. Latest developments in the SDPA family for solving large-scale SDPs. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications*, chapter 24, pages 687–714. Springer, NY, USA, 2012.

[31] M. Yamashita, K. Fujisawa, M. Fukuda, K. Nakata, and M. Nakata. Algorithm 925: Parallel solver for semidefinite programming problem having sparse Schur complement matrix. *ACM Trans. Math. Softw.*, 39(1), 2012. Article No.6.

[32] Makoto Yamashita and Kazuhide Nakata. Fast implementation for semidefinite programs with positive matrix completion. *Optim. Methods Softw.*, 2015. to appear.